



0x00000FF

ARGOS / DaeChoongCon@Zoom

2<sup>nd</sup>\_DCCCon\_0227\_2021



# 모든 웹을 이용한 안전한 웹 앱 손쉽게 설계하기

충남대학교 컴퓨터공학과  
ARGOS 허강준

# 발표자 소개



전에 넣었던 사진이 너무 못생겨서 다시 넣은 사진

허강준 @ ARGOS, 충남대학교

1인 기업 SIERRA-DELTA TECHNOLOGIES 대표

주로 사용하는 언어

C#, PHP, JavaScript @ Web

C/C++/C# @ Application

제 1회 대충콘 (19.07.06)

"Prepared Statement 올바르게 사용해보기"

Contact - knowledge@o.cnu.ac.kr / Twitter DM – 0x00000FF

# 오늘의 주제들

- 쿠키를 잘 구워서 대접하는 방법
  - 쿠키에 넣으면 좋은 / 좋지 않은 토픽들 / localStorage?
  - 어디에서 대접해야 할까? – SameSite Policy
  - 언제 대접 해야할까? – HttpOnly, Secure
- 잘 쓰면 정말 편해지는 헤더들
  - Downgrade Attack & MITM – HTTP Strict Transport Security(HSTS)
  - XSS - Content-Security-Policy (CSP), X-XSS-Protection
    - 리소스 무결성 검증 – Integrity Attribute
  - Clickjacking - X-Frame-Options
  - MIME Sniffing - X-Content-Type-Options
  - CORS – 알아두면 편리한 Allow-Access-Control 시리즈

# 오늘의 주제들

- 조금 주의해야 할 헤더들
  - Banner Grabbing – Server, X-Powered-By
  - Privacy Policy – Do Not Track(DNT)
- 사이트의 보안을 손쉽게 점검하기
  - 간단하게 점검하기 – immuniweb
  - 장비 동원하기 – ZAP, Burp Suite
- 사족: Web Application Firewall (WAF)
- 질문/답변

# 들어가기에 앞서...

- 모든 보안은 **입력을 신뢰하지 않는 것**에서 시작합니다.
- 모든 보안 메커니즘은 **언젠가는, 어느 식으로든** 파훼 될 수 있습니다.
- 오늘 다루는 내용은 많은 도움을 주겠지만 결코 **만능은 아닙니다**.

# 쿠키



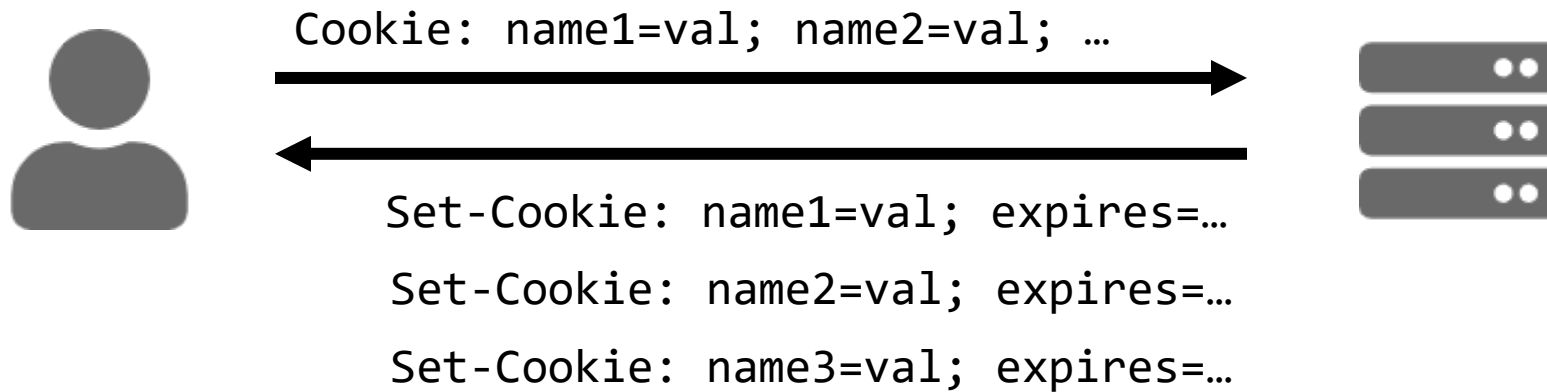
쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 쿠키

- HTTP는 상태가 없음(Stateless)을 전제로 한 프로토콜
- 현재의 연결은 이전이나 이후의 연결과 아무 관련 없음
- 상태(State) – 지금 연결되어 있나? 넌 누구냐? 아까 뭐했니?
- 상태를 표현하기 위한 방법?

# 쿠키

- 적당한 크기의 데이터를 통신할 때 주고받자!





# 쿠키 - 주 용도?

- 사용자 인증 (Authentication)
  - 세션쿠키, 인증토큰 등...
  - 로그인 할 때 [아이디 저장], [자동 로그인]
- 사용자 맞춤 서비스 제공
  - 최근에 방문한 게시판, 블로그, 사이트 등등...
  - 오늘 본 물건 목록
  - [24시간동안 표시하지 않습니다] 뭐 이런거...

# 쿠키 – 무엇을 넣고 / 넣지 말아야 할까?

- 쿠키도 결국 HTTP 패킷 안에 들어가는 데이터
  - ▶ 너무 많으면 통신 오버헤드가 커짐 (패킷 크기가 커짐!)
  - ▶ HTTP 통신은 생각보다 무겁다!
- 서버와 “상태” 를 유지해야 하는가? 를 생각
- “상태”를 유지해야 하지만 서버가 굳이 알 필요가 없다면?
  - ▶ localStorage에 저장

# 쿠키 – localStorage?

- localStorage는 키-값 형태의 데이터를 지원하는 전역 객체
- getItem(), setItem() 으로 설정하고 가져올 수 있음
- removeItem(), clear()로 값 제거
- 폭 넓은 호환성

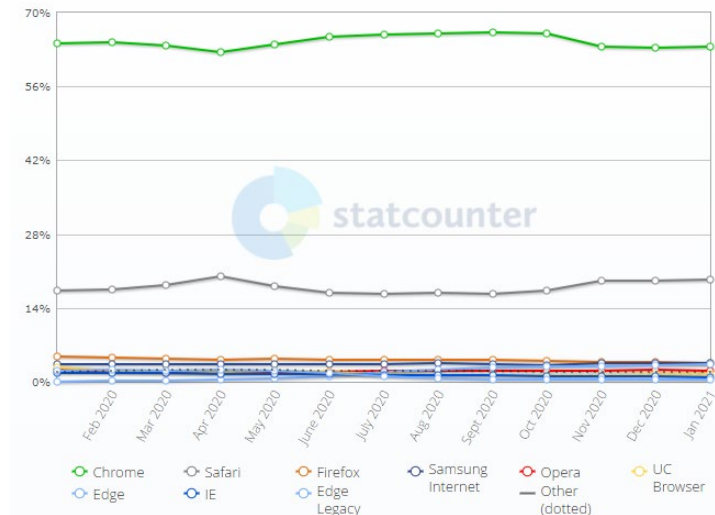
	🖥️						📱					
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox Android	Opera Android	iOS Safari	Samsung Internet
localStorage	4	12	3.5	8	10.5	4	≤ 37	18	4	11	3.2	1.0

Full support

Browser Market Share Worldwide

Jan 2020 - Jan 2021

Edit Chart Data



# 쿠키 – localStorage?

index.html

```
<button onclick="up_click()">숫자올리기!</button>
<button onclick="count_clear()">초기화</button>
<button onclick="check()">얼마나 눌렀나?</button>

<script>
  function up_click() {
    localStorage.clicks++;
  }

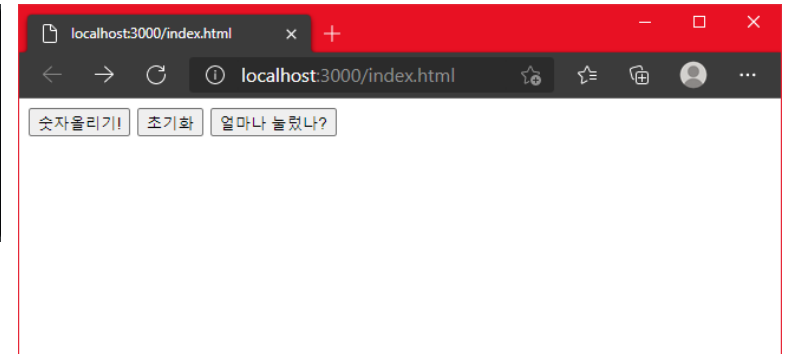
  function count_clear() {
    localStorage.clicks = 0;
  }

  function check() {
    location.href = "check.html";
  }

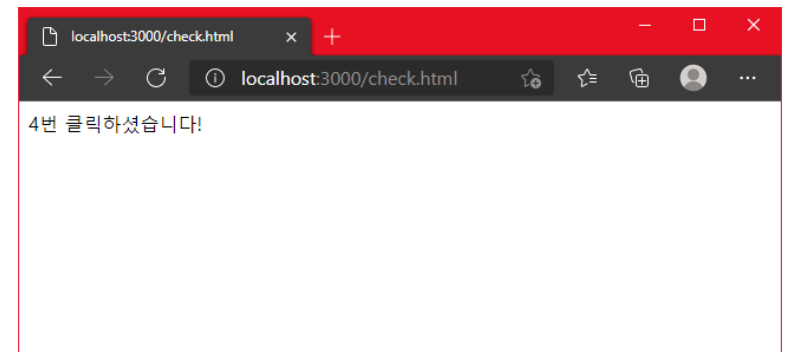
  (function () {
    if (!localStorage.clicks) {
      localStorage.clicks = 0;
    }
  })();
</script>
```

check.html

```
<script>
  document.write(
    localStorage.clicks + "번 클릭하셨습니다!");
</script>
```



4번 클릭 후 확인



# 쿠키 – localStorage?

- 암호화 미지원 – 중요한 데이터는 넣지 말아야 함!
- 특정 세션동안에만 유지되어야 한다면 – sessionStorage
- localStorage는 따로 삭제하지 않는 이상 영원히 보관됨
- 사용법은 sessionStorage나 localStorage나 동일

# 쿠키 – 어디에서 쿠키를 대접할까?

- 쿠키는 특정한 조건을 만족할 때에만 전송됨
- 만료된 쿠키 – Expire 시간이 지난 쿠키는 전송되지 않고 삭제
- 도메인에 관한 조건들: SameSite, Domain, Path

쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 쿠키 – SameSite

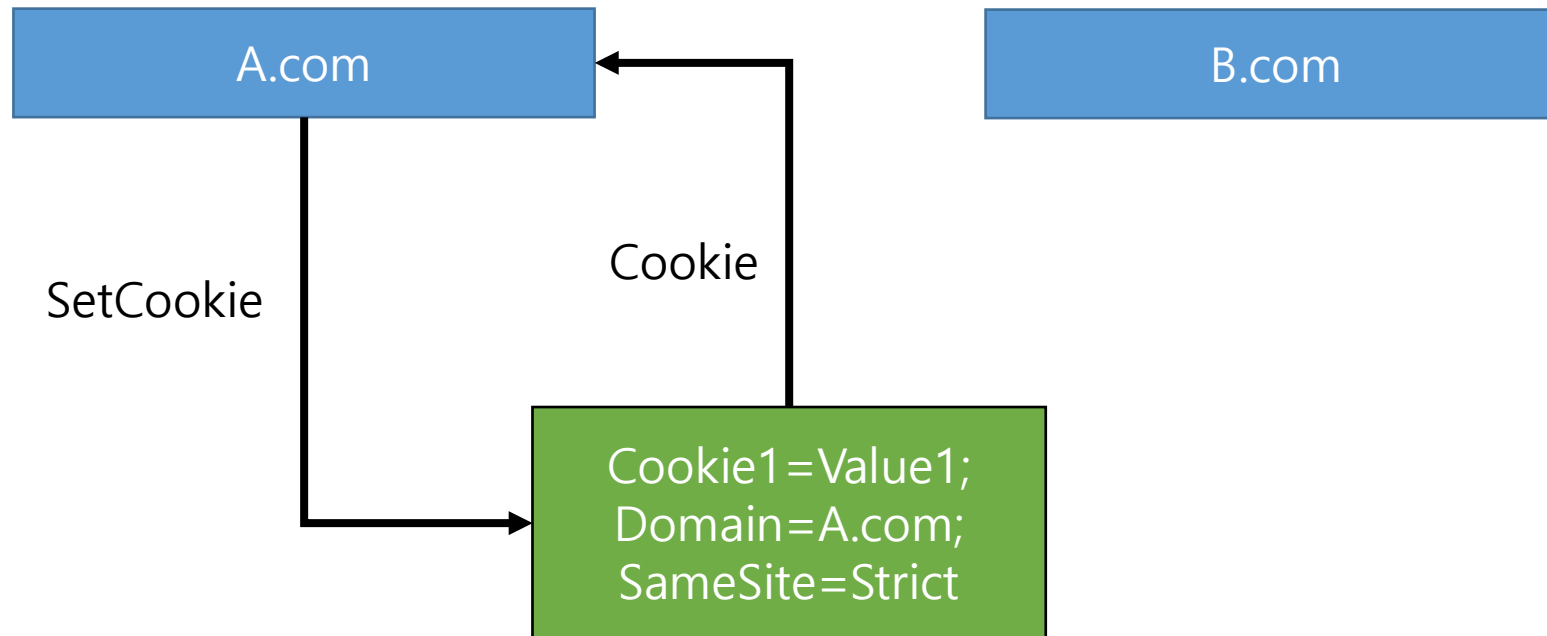
- 쿠키는 도메인에 따라 식별됨
- SameSite Policy: 동일 Domain/프로토콜에서만 사용?
- 인증 관련 쿠키의 무분별한 전송은 CSRF 공격에 취약

Strict	Cross-Origin 비허용
Lax	Strict와 동일하나 외부에서 특정 조건 하에서 링크 들어올 때만 허용 (Chrome 80부터 기본값)
None	"Secure" 조건 하에 Cross-Origin 허용 (원래 기본값)

# 쿠키 – SameSite = Strict

- 타 도메인에서 쿠키 전송 안함

e.g.) A.com의 쿠키는 B.com에서 전송되지 않음

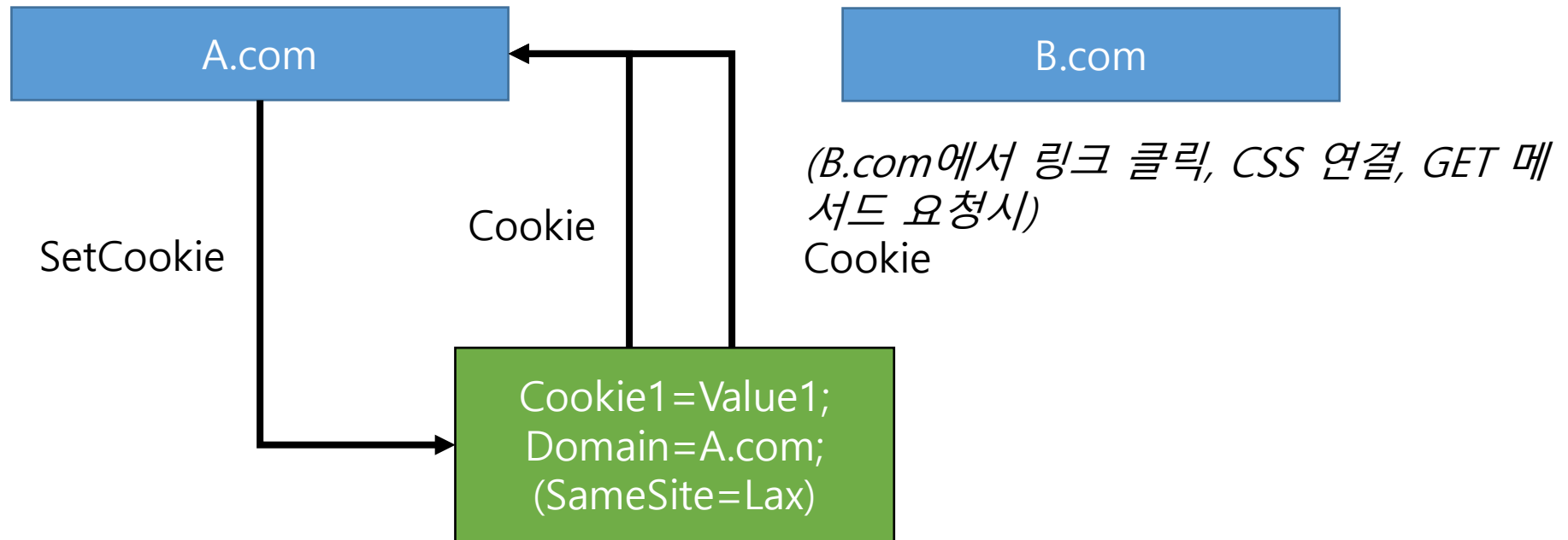


쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------



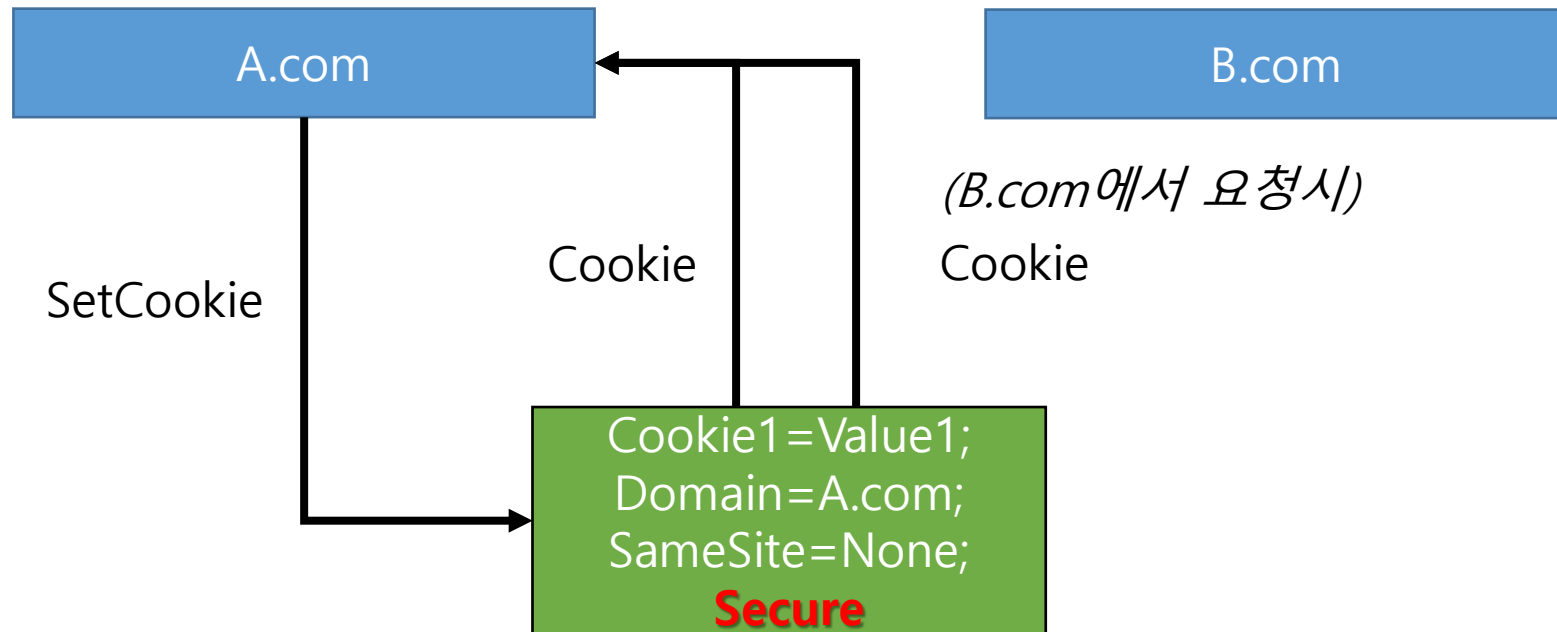
# 쿠키 – SameSite = Lax

- 일반적인 경우 Strict와 동일
- 단 a href, link href, GET 메서드를 이용한 Form 요청은 예외



# 쿠키 – SameSite = None

- 타 도메인에서 쿠키 전송 허용
- 단 Secure 적용된 쿠키 한정 (그 외에는 전송하지 않음)



# 쿠키 – 언제 대접할까?

- 아무런 설정이 없는 경우 쿠키는 모든 영역에서 접근 가능
  - 스크립트에서 `document.cookie`로 접근 가능하다
  - 즉, XSS에 취약하다!
- 
- 세션 쿠키나 인증 토큰은 탈취 당하면 골치가 정말 아프다
  - 특정 상황에서만 전송할 수는 없을까?

# 쿠키 – HttpOnly와 Secure

- HttpOnly는 HTTP 통신이 발생할 때만 전송  
Cookie1=value; **HttpOnly**
- Secure는 HTTPS 통신이 발생할 때만 전송  
Cookie2=value; **Secure**

# 쿠키 – HttpOnly와 Secure

- 쿠키 세개를 설정하는 웹사이트(Http)

```
Set-Cookie: sample_cookie_none=hello%21  
Set-Cookie: sample_cookie_ho=hello%20HttpOnly%21; HttpOnly  
Set-Cookie: sample_cookie_sec=hello%20Secure%21; secure ⚠
```

이 Set-Cookie는 "Secure" 특성이 있지만 보안 연결을 통해 수신되지 않았기 때문에 차단되었습니다.

- 콘솔에서 접근하면?

```
> document.cookie  
< "sample_cookie_none=hello%21"
```

- sample\_cookie\_none 외에는 document.cookie 로 접근 불가능

# 쿠키 – HttpOnly와 Secure

- 다시 접속을 시도해보면....

```
Cookie: sample_cookie_none=hello%21; sample_cookie_ho=hello%20HttpOnly%21
```

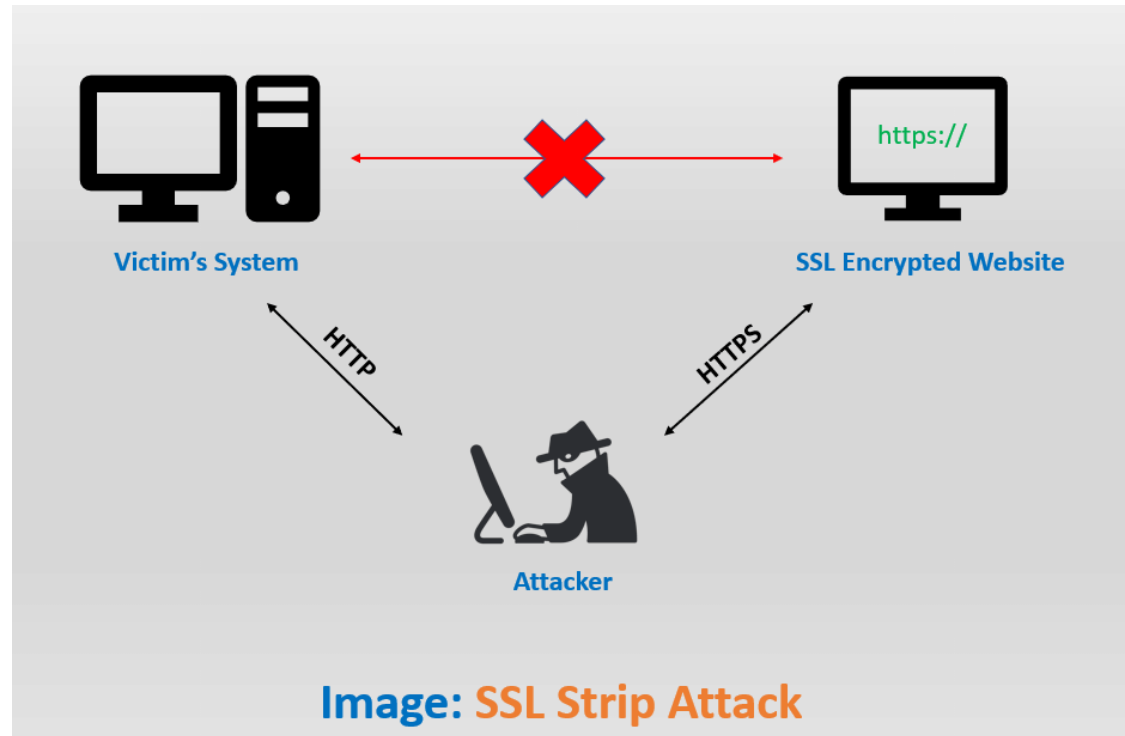
- Secure가 붙었던 sample\_cookie\_sec는 전송되지 않는다
- **Https 연결에서만 전송**되기 때문에 설정되지 않았기 때문
- 다른 https 페이지에서 설정되었더라도 전송되지 않는다

# Protocol Downgrade Attack / Man-In-The-Middle

- HTTP는 암호화 되지 않은 평문통신
- 중간에 패킷만 잡아낸다면 전부 볼 수 있다!
- 물론 조작도 가능

쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# Protocol Downgrade Attack / Man-In-The-Middle



<https://cheapsslsecurity.com/blog/what-is-hsts-a-brief-overview-of-hsts-technology/>



# 언제나 안전하게 – HTTP Strict Transport Security

- 서버가 브라우저에 HTTPS로 접속하도록 강제

- 사용법:

Strict-Transport-Security: max-age=[0-9]\*; (includeSubdomains|preload)

- \* max-age: 브라우저가 HTTPS 접속을 하도록 기억하는 시간 (초, 보통 1달~1년)
- \* includeSubdomains: 서브도메인까지 적용
- \* preload: preload list에 등록되도록 설정

# 언제나 안전하게 – HTTP Strict Transport Security

## • Preload List?

Enter a domain:

example.com

Check HSTS preload status and eligibility

**Information**

This form is used to submit domains for inclusion in Chrome's [HTTP Strict Transport Security \(HSTS\)](#) preload list. This is a list of sites that are hardcoded into Chrome as being HTTPS only.

Most major browsers (Chrome, [Firefox](#), Opera, Safari, [IE 11 and Edge](#)) also have HSTS preload lists based on the Chrome list. (See the [HSTS compatibility matrix](#).)

**Submission Requirements**

If a site sends the `preLoad` directive in an HSTS header, it is considered to be requesting inclusion in the preload list and may be submitted via the form on this site.

In order to be accepted to the HSTS preload list through this form, your site must satisfy the following set of requirements:

1. Serve a valid **certificate**.
2. **Redirect** from HTTP to HTTPS on the same host, if you are listening on port 80.
3. Serve all **subdomains** over HTTPS.
  - In particular, you must support HTTPS for the `www` subdomain if a DNS record for that subdomain exists.
4. Serve an **HSTS header** on the base domain for HTTPS requests:
  - The `max-age` must be at least 31536000 seconds (1 year).
  - The `includeSubDomains` directive must be specified.
  - The `preload` directive must be specified.
  - If you are serving an additional redirect from your HTTPS site, that redirect must still have the HSTS header (rather than the page it redirects to).

1. Serve a valid **certificate**.
2. **Redirect** from HTTP to HTTPS on the same host, if you are listening on port 80.
3. Serve all **subdomains** over HTTPS.
  - In particular, you must support HTTPS for the `www` subdomain if a DNS record for that subdomain exists.
4. Serve an **HSTS header** on the base domain for HTTPS requests:
  - The `max-age` must be at least 31536000 seconds (1 year).
  - The `includeSubDomains` directive must be specified.
  - The `preload` directive must be specified.
  - If you are serving an additional redirect from your HTTPS site, that redirect must still have the HSTS header (rather than the page it redirects to).

# XSS (Cross Site Scripting)


- 사이트에 개발자가 의도하지 않은 스크립트를 삽입하는 공격
- 정말 다양한 공격 경로들  
<script>, <link>, 이벤트 핸들러...
- 또 정말 다양한 꼼수들과 진화하는 필터들  
그리고 진화하는 꼼수들...
- 핵심은 신뢰할 수 없는 스크립트의 무단실행 방지


# 단순한 방법이지만... - X-XSS-Protection


- 이름 그대로 XSS를 방어해주는 헤더
- 브라우저에서 XSS가 감지되면 렌더링 중단 or 스크립트 제거
- 사용법:  
`X-XSS-Protection: (0|1); (mode=|report=)`
- `mode=block` 일 경우 스크립트 제거
- `report=url` 을 지정할 경우 url로 XSS 공격 정보 전송


# 단순한 방법이지만... - X-XSS-Protection

- 단점: 비표준이다

|   | <div></div>  |         |         |                   |        |        | <div></div>     |                |                 |               |            |                  |
|---|--------------|---------|---------|-------------------|--------|--------|-----------------|----------------|-----------------|---------------|------------|------------------|
|   | Chrome       | Edge    | Firefox | Internet Explorer | Opera  | Safari | WebView Android | Chrome Android | Firefox Android | Opera Android | iOS Safari | Samsung Internet |
| X-XSS-Protection  | No<br>4 — 78 | 12 — 17 | No      | 8                 | ? — 65 | Yes    | No              | ? — 78         | No              | ? — 56        | Yes        | Yes              |

 Full support

 No support

 Non-standard. Expect poor cross-browser support.

# 단순한 방법이지만... - X-XSS-Protection

- 단점2: Stored XSS를 못 막는다(!)  
Request에서 스크립트를 감지하는 방식이기 때문
- 단점3: 요즘들어 모던 브라우저에서 지원이 안된다  
옛날 브라우저나 IE 지원하는거 아니면...

- Chrome has [removed their XSS Auditor](#)
- Firefox has not, and [will not implement X-XSS-Protection](#)
- Edge has [retired their XSS filter](#)

This means that if you do not need to support legacy browsers, it is recommended that you use [Content-Security-Policy](#) without allowing [unsafe-inline](#) scripts instead.

# 단순한 방법이지만... - X-XSS-Protection

- 결론

보안용으로 쓰지 마세요!

(보안감사 같은 데서 필요하면 장식용으로 달아두세요...)

# 확실하고 깔끔한 방법 - Content-Security-Policy

- 사이트 내의 각 요소(embed, img, iframe, script, link...)에 대해 브라우저에서의 원본 위치 검증을 요청하는 헤더

- 사용법:

Content-Security-Policy: <src-type> (self|'blah'|'nonce-blah')\*; ...

|    |          |           |      |
|----|----------|-----------|------|
| 쿠키 | 헤더(넣을 것) | 헤더(조심할 것) | 점검도구 |
|----|----------|-----------|------|



# 확실하고 깔끔한 방법 - Content-Security-Policy

- src-type?

|             |  |
|-------------|--|
| default-src | 기본값 (따로 지정되지 않았을때의 정책)                 |
| script-src  | 스크립트 (<script>)                        |
| style-src   | 스타일시트 (CSS)                            |
| img-src     | 이미지                                    |
| connect-src | AJAX, 웹소켓, fetch 등등                    |
| font-src    | 폰트 (@font-face 등으로 정의되는 것들)            |
| object-src  | <object>, <embed>, <applet> (주로 플러그인들) |
| media-src   | <audio>, <video>와 같은 미디어들              |
| child-src   | <iframe> 같은 프레임들                       |

# 확실하고 깔끔한 방법 - Content-Security-Policy

- 설정 가능한 위치들

|                     |                                    |
|---------------------|------------------------------------|
| *                   | 전부 허용 (제한 안함)                      |
| 'none'              | 전부 제한 (아예 허용 안함)                   |
| 'self'              | 동일 도메인, 포트, 프로토콜에서 허용              |
| data:               | Data URI로 지정된 리소스 허용               |
| abc.example.com     | abc.example.com의 리소스 허용            |
| *.example.com       | example.com의 전체 서브도메인까지 허용         |
| https://example.com | HTTPS 프로토콜 하에서 example.com의 리소스 허용 |
| https:              | HTTPS 프로토콜이면 허용                    |
| 'nonce-'            | nonce를 가지고 있는 리소스는 허용              |

# 확실하고 깔끔한 방법 - Content-Security-Policy

- nonce?
- href나 src로 연결되지 않고 inline으로 넣고 싶을 땐 어떻게?
- “난수” 를 가지고 있는 요소를 허용

Example)

Content-Security-Policy: script-src 'nonce-ac341f9c8a76e';

`<script nonce="ac341f9c8a76e"> </script>` `<!-- 이걸 실행됨 -->`

`<script></script>` `<!-- 이걸 실행 안됨 -->`

# 확실하고 깔끔한 방법 - Content-Security-Policy

- “난수” 생성은 자유롭게, 하지만!
- 공격하는 측이 예측하기 힘든 값으로 해야함

Example)

```
$rand = random_string(32); // 256비트 랜덤스트링 생성  
$rand = sha256($rand);    // 해시로 만들기
```

```
// 이후 Content-Security-Policy에 nonce-$rand 식으로 삽입
```

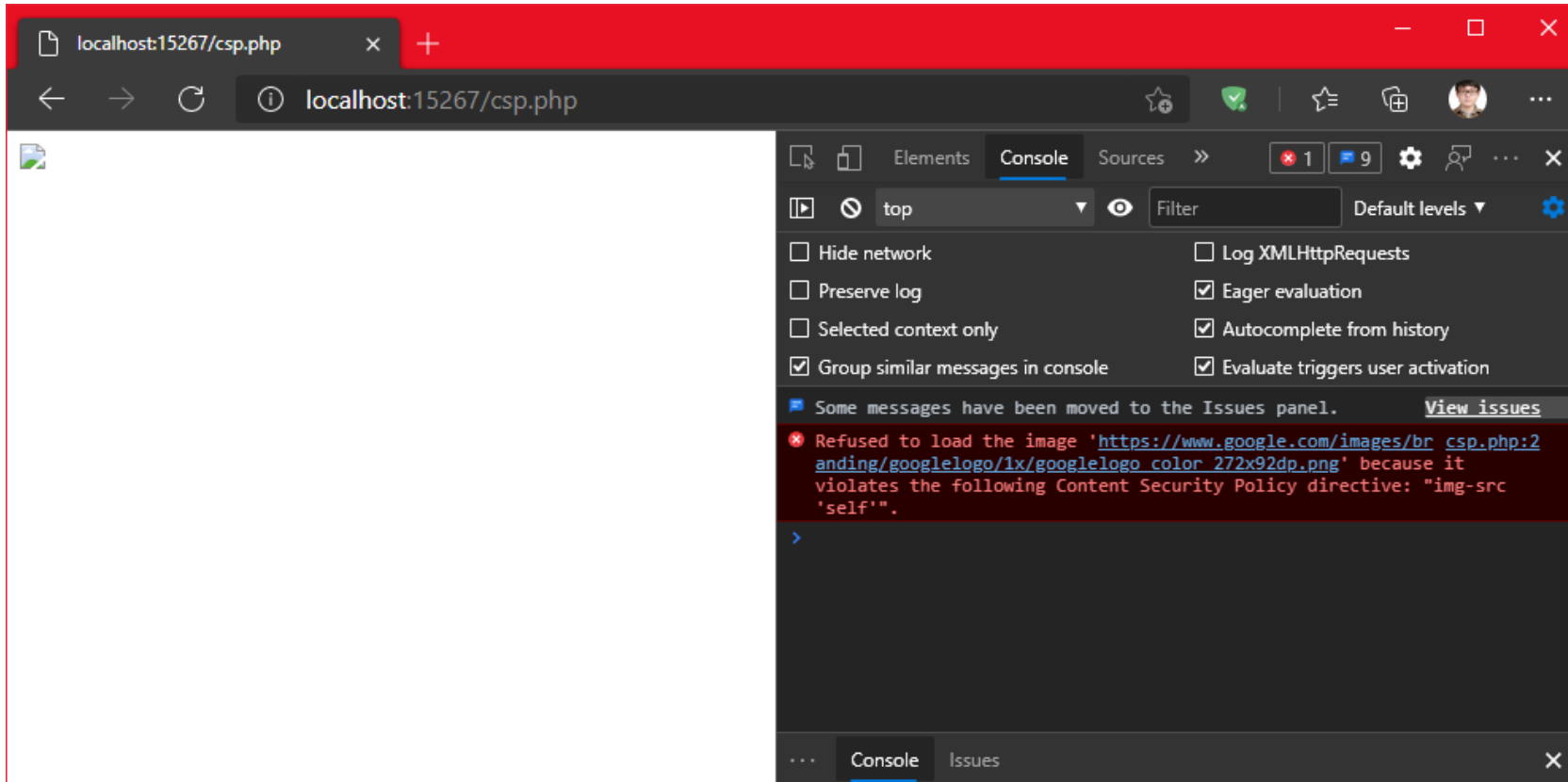
쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 확실하고 깔끔한 방법 - Content-Security-Policy

- img-src: 'self' 한 뒤 구글 로고를 불러오면?

```
csp.php
1  <?
2  |    header("Content-Security-Policy: image-src 'self'");
3  |  ?>
4  |
5  |    
```

# 확실하고 깔끔한 방법 - Content-Security-Policy



쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 확실하고 깔끔한 방법 - Content-Security-Policy

```
content-security-policy: default-src 'self'; connect-src 'self'; img-src 'self' *.hellodd.com
data:; frame-src *.youtube.com kapi.kakao.com 'nonce-90275b506023932804f31d7ed473299d38b61
f95e42d7a0594fdce38b2d2ef00975c16102570aa296b557a7fd370d9397c2bd36ecd583b35fa77a2913ffb4a3
a'; object-src *.youtube.com 'nonce-90275b506023932804f31d7ed473299d38b61f95e42d7a0594fdce
38b2d2ef00975c16102570aa296b557a7fd370d9397c2bd36ecd583b35fa77a2913ffb4a3a'; script-src 's
elf' 'unsafe-eval' 'nonce-90275b506023932804f31d7ed473299d38b61f95e42d7a0594fdce38b2d2ef00
975c16102570aa296b557a7fd370d9397c2bd36ecd583b35fa77a2913ffb4a3a'
```

# 확실하고 깔끔한 방법 - Content-Security-Policy

## CSP Browser Support

Content Security Policy is supported by all the major modern browsers, and has been for many years. It is not supported in Internet Explorer.

### Chrome

Content-Security-Policy **CSP Level 3** - Chrome 59+ Partial Support  
Content-Security-Policy **CSP Level 2** - Chrome 40+ Full Support Since January 2015  
Content-Security-Policy **CSP 1.0** - Chrome 25+  
X-Webkit-CSP **Deprecated** - Chrome 14-24

### FireFox

Content-Security-Policy **CSP Level 3** - Firefox 58+ Partial Support  
Content-Security-Policy **CSP Level 2** - FireFox 31+ *Partial Support* since July 2014  
Content-Security-Policy **CSP 1.0** - FireFox 23+ Full Support  
X-Content-Security-Policy **Deprecated** - FireFox 4-22

### Safari

Content-Security-Policy **CSP Level 2** - Safari 10+  
Content-Security-Policy **CSP 1.0** - Safari 7+  
X-Webkit-CSP **Deprecated** - Safari 6

### Edge

Content-Security-Policy **CSP Level 3** - Edge 79+ Partial Support  
Content-Security-Policy **CSP Level 2** - Edge 15+ Partial, 76+ Full  
Content-Security-Policy **CSP 1.0** - Edge 12+

### Internet Explorer

X-Content-Security-Policy **Deprecated** - IE 10-11 support **sandbox** only



# 확실하고 깔끔한 방법 - Content-Security-Policy

- 정말 좋은 물건인데...
- 90% 이상의 국내 사이트들은 채택하지 않고 있는 헤더

```
▼ 일반
요청 URL: https://cafe.naver.com/joonggonara
요청 메서드: GET
상태 코드: 200
원격 주소: 210.89.168.34:443
참조 페이지 정책: strict-origin-when-cross-origin

▼ 응답 헤더
cache-control: no-cache
content-type: text/html; charset=MS949
date: Thu, 25 Feb 2021 05:45:30 GMT
expires: Thu, 01 Jan 1970 00:00:00 GMT
p3p: CP="ALL CURa ADMa DEVa TAIa OUR BUS IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA POL HEA PRE LOC OTC"
referrer-policy: unsafe-url
server: nfront
set-cookie: [redacted]; Path=/; HttpOnly
vary: Accept-Encoding, User-Agent
x-xss-protection: 1; mode=block
```

쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 확실하고 깔끔한 방법 - Content-Security-Policy

- 참 개발자라면 열심히 사용해주시길
- 아주 조금만 더 부지런해지면 돼요
- 서버 자체가 털리지 않는 이상 정말 안전해져요...

CSP 끝!

# 클릭재킹(Clickjacking)?



© Yasunobu Yamauchi, 罪罪罪罪罪罪罪罪罪, Ep 7. <Hijack>

# 클릭재킹(Clickjacking)?

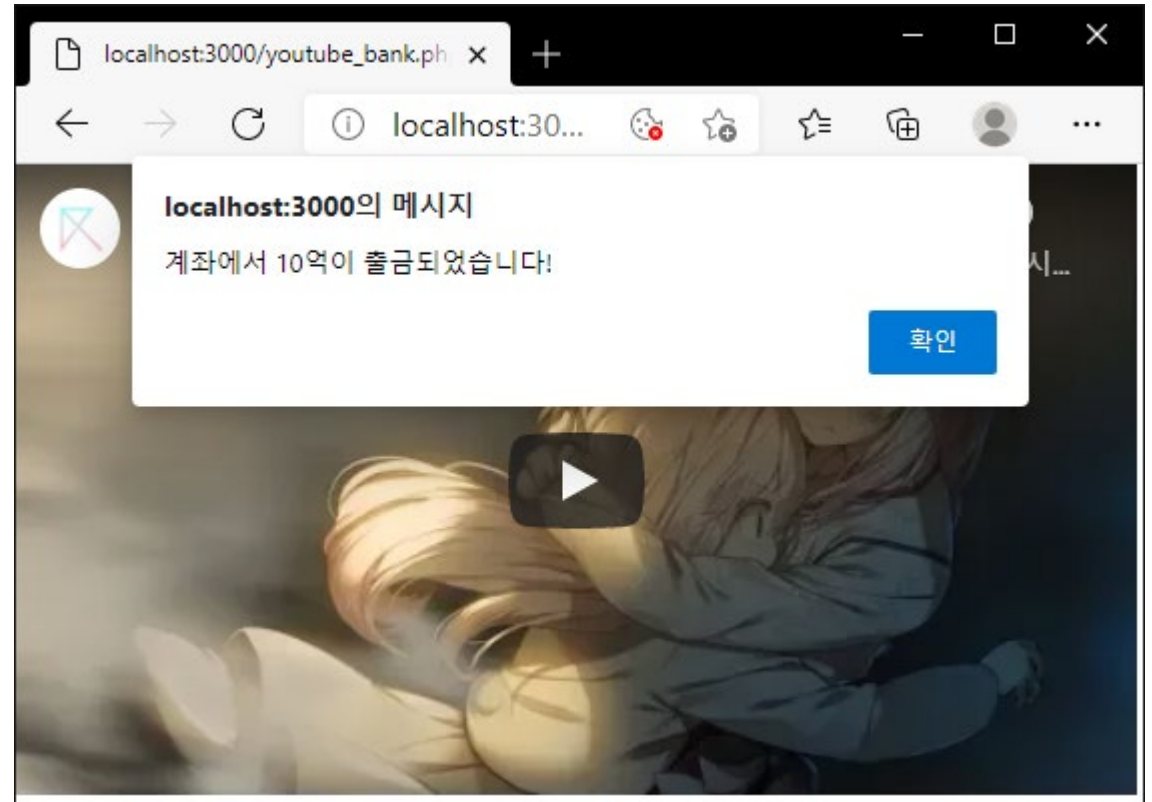
- UI Redressing Attack의 일종
- 클릭하지 않은 곳을 클릭한 것처럼 사이트 변조  
e.g.) [재생] 버튼을 눌렀지만 광고가 뜬다거나...
- 보통 타 서버의 페이지나 파일을 불러올 수 있는 방법을 사용  
<iframe>, <object>, <embed>, ...

# 클릭재킹(Clickjacking)?

- 사용자를 본인의 서버로 접속하도록 유도(피싱 등)
- 공격 대상 페이지를 iframe 안에 표시
- 그 위에 올려 공격자가 원하는 스크립트를 실행하도록 구성

# 클릭재킹(Clickjacking)?

```
youtube_bank.php
1 <style>
2   * { margin: 0; padding: 0 }
3   .clickjack {
4     position: absolute;
5     top: 0;
6     left: 0;
7     width: 100%;
8     height: 100%;
9     cursor: pointer;}
10 </style>
11
12 <iframe width="560" height="315" src="https://www.youtube.com/embed/gx..."
13   frameborder="0"
14   allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share"
15 <div class="clickjack"> </div>
16 <script>
17   document.querySelector(".clickjack")
18     .addEventListener("click",
19     () => alert("계좌에서 10억이 출금되었습니다!"))
20 </script>
```



# 프레임 허용 정책 – X-Frame-Options

- 브라우저가 <frame>, <iframe>, <embed>, <object>의 원본에 대해 렌더링 할지에 대해 결정
- 공격자가 내 사이트의 콘텐츠를 삽입할 수 없게 됨

- 사용법:

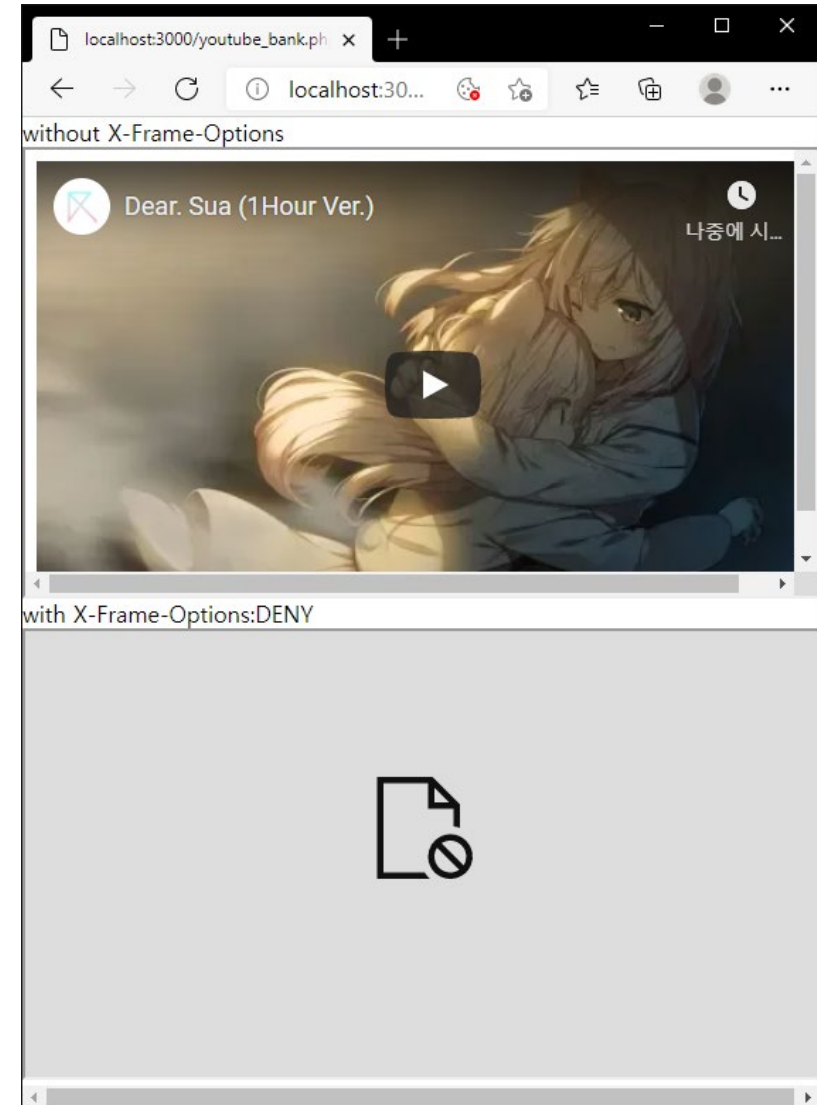
X-Frame-Options: (DENY|SAMEORIGIN)

- DENY: 해당 페이지는 무조건 프레임 안에서 작동하지 않음
- SAMEORIGIN: 현재 보고 있는 페이지와 동일한 Origin에서만 프레임 허용

# 프레임 허용 정책 – X-Frame-Options

```
show_youtube_allow.php
show_youtube_deny.php
youtube_bank.php
```

```
Server responded with a status of 300 ( )
✖ Refused to display 'http://localhost:3000/show_youtube_youtube_bank.php:1_deny.php' in a frame because it set 'X-Frame-Options' to 'deny'.
```





# 프레임 허용 정책 – X-Frame-Options

- 프레임의 origin을 체크한다는 면에서 CSP와 결합 가능
- CSP의 frame-ancestors – 프레임을 넣은 사이트 검증
- 내 사이트가 다른 사이트에 프레임으로 삽입되고 싶지 않다면?
- 다음과 같이 헤더를 구성

X-Frame-Options: deny;

Content-Security-Policy: ...; frame-ancestors: 'self' \*.example.com ...; ...

# MIME Sniffing

- 이미지가 아니지만 <img>로 불러오는 경우?
- 서버는 image라고 MIME을 보내지만 실제 파일은 아님
- 브라우저는 파일을 분석하여 추측 실행하는 기능이 있음

# 서버를 믿어주세요! – X-Content-Type-Options

- 브라우저의 추측 실행을 막는 헤더
- 서버에서 전송한 MIME(Content-Type)과 실제 파일이 불일치할 경우 서버의 값을 따름
- 사용법:  
X-Content-Type-Options: nosniff

# 보너스: 리소스 무결성 검증 - Integrity Attribute

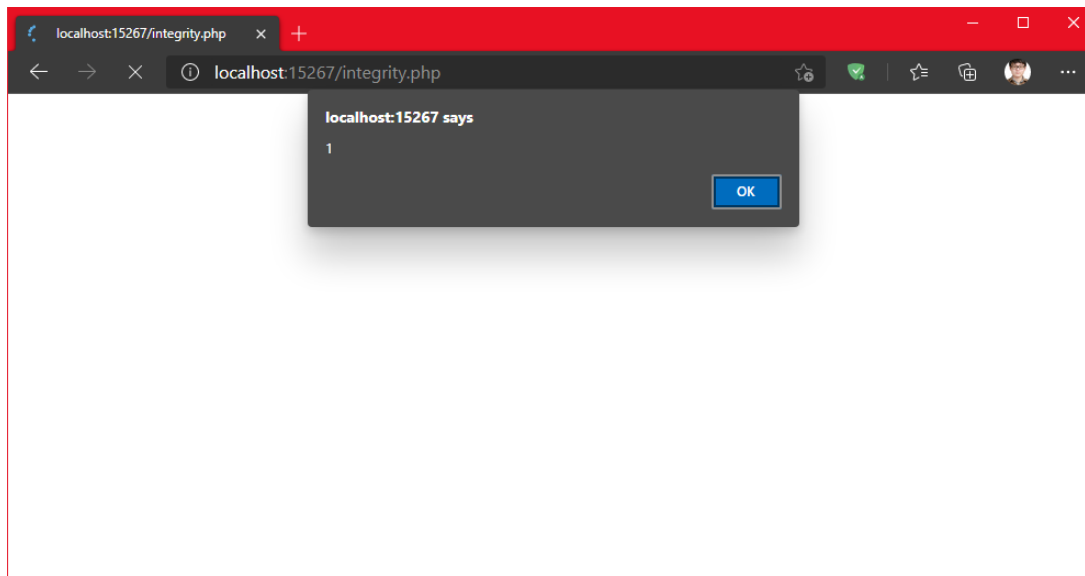
- 페이지 외적으로 로딩되는 리소스는 변조될 수 있음
- HTML Attribute: integrity
- SRI 해시를 생성하여 값으로 사용
- 스크립트, CSS에 적용 가능

Example)

```
<script src="..." integrity="sha384-BASE64BLAHBLAH=="></script>
```

# 보너스: 리소스 무결성 검증 - Integrity Attribute

```
integrity.php
1 <!-- alert(1); -->
2 <script src="script_ok.js" integrity="sha384-dnux3uAPxaf+IhCrFG1D/XVNzP1XLNcn3Pe3jyxouEAoot5kfwC5u8rMwNhE5oi"></script>
3 <!-- alert(2); -->
4 <script src="script_no.js" integrity="sha384-dnux3uAPxaf+IhCrFG1D/XVNzP1XLNcn3Pe3jyxouEAoot5kfwC5u8rMwNhE5oi"></script>
```



```
✖ Failed to find a valid digest in the 'integrity' attribute for resource 'http://localhost:15267/script_no.js' with computed SHA-256 integrity '4+wJxJ8EX2pPgiQ/zzorRCP/My8UOWxQIRHmkPA1PeU='. The resource has been blocked. integrity.php:1
```

# 보너스: 리소스 무결성 검증 - Integrity Attribute

- SRI(Subresource Integrity) Hash를 만드는 법?

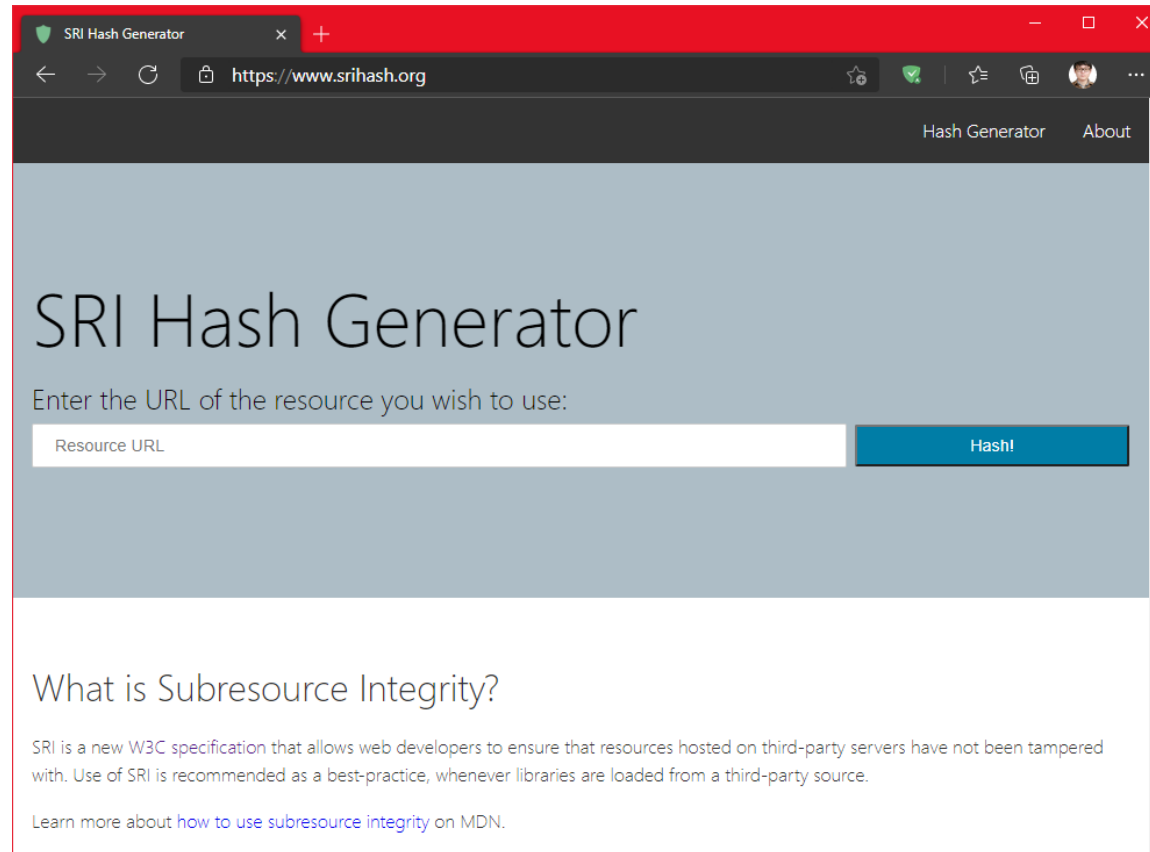
## NOTE

Digests may be generated using any number of utilities. [OpenSSL](#), for example, is quite commonly available. The example in this section is the result of the following command line:

```
echo -n "alert('Hello, world.');" | openssl dgst -sha384 -binary | openssl base64 -A
```

- <https://www.srihash.org/>

# 보너스: 리소스 무결성 검증 - Integrity Attribute



# Cross Origin Resource Sharing (CORS)

- 때로는 API를 외부에 공개할 필요가 있다
- 하지만...

```
✖ Access to XMLHttpRequest at '...' (인덱스):1  
from origin '...' has been blocked by CORS policy: No  
'Access-Control-Allow-Origin' header is present on the requested resource.
```

- Cross-Origin XMLHttpRequest는 보안상 금지되어 있음



# 들어가요 될까요? – Preflight

- 다른 서비스들은 어떻게 API를 제공하는 걸까?
- 에러메세지를 다시 자세히 보면...

```
⊗ Access to XMLHttpRequest at ' ' (인덱스):1  
from origin ' ' has been blocked by CORS policy: No  
'Access-Control-Allow-Origin' header is present on the requested resource.
```

# 들어가요 될까요? – Preflight

- 브라우저는 CORS 요청 전 Preflight 요청을 먼저 보냄
- 현재 CORS 요청이 유효한지 서버에 정보 요청

Access-Control-Allow-Origin	유효한 Origin인가? (요청을 시도하는 도메인)
Access-Control-Allow-Credentials	요청에 Credential을 포함할 수 있는가?
Access-Control-Allow-Methods	요청에 사용할 수 있는 Method인가?
Access-Control-Allow-Headers	요청에 사용할 수 있는 Header인가?

# 주고 싶은 사람에게만 주는 – Access-Control-Allow-Origin

- Access-Control-Allow-Origin
- Cross-Origin 요청을 할 수 있는 사이트를 지정

- 사용법:

Access-Control-Allow-Origin: [www.naver.com](http://www.naver.com) // 네이버에서 호출 가능

<or>

Access-Control-Allow-Origin: blog.patche.me // 블로그에서 호출 가능

# 주고 싶은 사람에게만 주는 – Access-Control-Allow-Origin

- 여러 곳에서 (혹은 아무 곳에서나) 요청할 때는 어떻게?

Access-Control-Allow-Origin: \*

- 하지만 Credential이 필요한 요청이라면...

```
> jQuery.ajax({
  url: "http://localhost:3000/cors.php",
  xhrFields: { withCredentials: true },
});
< {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}
```

✖ Access to XMLHttpRequest at 'http://localhost:3000/cors.php' (인덱스):1 from origin 'https://jquery.com' has been blocked by CORS policy: The value of the 'Access-Control-Allow-Origin' header in the response must not be the wildcard '\*' when the request's credentials mode is 'include'. The credentials mode of requests initiated by the XMLHttpRequest is controlled by the withCredentials attribute.

# 주고 싶은 사람에게만 주는 – Access-Control-Allow-Origin

- 백엔드에서 허용할 URL 리스트를 만들어놓고 검사하거나
- 현재 요청한 Origin을 그대로 헤더에 적용

```
$origin = $_SERVER["HTTP_ORIGIN"];  
header("Access-Control-Allow-Origin: $origin");
```

// 이제 될까?

# 주고 싶은 사람에게만 주는 – Access-Control-Allow-Origin

- 아니오

```
> jQuery.ajax({  
  url: "http://localhost:3000/cors.php",  
  xhrFields: { withCredentials: true },  
});  
< {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}
```

✖ Access to XMLHttpRequest at '<http://localhost:3000/cors.php>' (인덱스):1 from origin '<https://jquery.com>' has been blocked by CORS policy: The value of the 'Access-Control-Allow-Credentials' header in the response is '' which must be 'true' when the request's credentials mode is 'include'. The credentials mode of requests initiated by the XMLHttpRequest is controlled by the withCredentials attribute.

# 인증이 필요하다면 – Access-Control-Allow-Credentials

- Access-Control-Allow-Credentials
- 요청에 Credentials(인증에 필요한 정보)를 포함할 수 있는가
- 허용하려면 true를 지정 후 전송

- 사용법:

Access-Control-Allow-Credentials: true

# 인증이 필요하다면 – Access-Control-Allow-Credentials

- 아까 전 소스를 살짝 수정

```
$origin = $_SERVER["HTTP_ORIGIN"];  
header("Access-Control-Allow-Origin: $origin");  
header("Access-Control-Allow-Credentials: true"); // 헤더 추가  
  
// 이제 될까?
```

쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------



# 인증이 필요하다면 – Access-Control-Allow-Credentials

```
> jQuery.ajax({
  url: "http://localhost:3000/cors.php",
  xhrFields: { withCredentials: true },
  success: response => console.log(response)
});
{readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}
hi VM1063:4
```



# 특정 메서드만 – Access-Control-Allow-Methods

- 어떤 요청은 특정 메서드 만을 써야할 경우가 있다 (DELETE 같은거)
- 보통 GET, POST는 특별한 조치 없이도 사용 가능

# 특정 메서드만 – Access-Control-Allow-Methods

```
hi VM42:2
> jQuery.ajax({
  url: "http://localhost:3000/cors.php",
  method: "GET",
  xhrFields: { withCredentials: true },
  success: response => console.log(response)
});
< {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}

hi VM51:5
> jQuery.ajax({
  url: "http://localhost:3000/cors.php",
  method: "POST",
  xhrFields: { withCredentials: true },
  success: response => console.log(response)
});
< {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}

hi VM60:5
```

```
> jQuery.ajax({
  url: "http://localhost:3000/cors.php",
  method: "PUT",
  xhrFields: { withCredentials: true },
  success: response => console.log(response)
});
< {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}

✖ Access to XMLHttpRequest at 'http://localhost:3000/cors.php' (인덱스):1
  from origin 'https://jquery.com' has been blocked by CORS policy: Method
  PUT is not allowed by Access-Control-Allow-Methods in preflight response.

✖ ▶ PUT http://localhost:3000/cors.php jquery-1.11.3.js:9664
  net::ERR_FAILED

> jQuery.ajax({
  url: "http://localhost:3000/cors.php",
  method: "DELETE",
  xhrFields: { withCredentials: true },
  success: response => console.log(response)
});
< {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}

✖ Access to XMLHttpRequest at 'http://localhost:3000/cors.php' (인덱스):1
  from origin 'https://jquery.com' has been blocked by CORS policy: Method
  DELETE is not allowed by Access-Control-Allow-Methods in preflight
  response.

✖ ▶ DELETE http://localhost:3000/cors.php jquery-1.11.3.js:9664
  net::ERR_FAILED
```

# 특정 메서드만 – Access-Control-Allow-Methods

- 의미를 명확히 하기 위해 특정 메서드를 사용할 경우가 있음
- PUT이나 DELETE만 허용해야 할 경우에는?
- 혹은 전부 허용하고 싶을 때는?

# 특정 메서드만 – Access-Control-Allow-Methods

- Access-Control-Allow-Methods
- CORS 요청에서 사용할 수 있는 HTTP 메서드를 지정

- 사용법:

Access-Control-Allow-Methods: GET, PUT // GET과 PUT만 허용

Access-Control-Allow-Methods: \* // 전부 다 허용

# 특정 메서드만 – Access-Control-Allow-Methods

- 아까 전 소스를 한번 더 수정

```
$origin = $_SERVER["HTTP_ORIGIN"];  
header("Access-Control-Allow-Origin: $origin");  
header("Access-Control-Allow-Credentials: true");  
header("Access-Control-Allow-Methods: DELETE"); // DELETE를 허용해보자
```

# 특정 메서드만 – Access-Control-Allow-Methods

```
> jQuery.ajax({
  url: "http://localhost:3000/cors.php",
  method: "DELETE",
  xhrFields: { withCredentials: true },
  success: response => console.log(response)
});
{readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}
hi
```

[VM104:5](#)

```
> jQuery.ajax({
  url: "http://localhost:3000/cors.php",
  method: "PUT",
  xhrFields: { withCredentials: true },
  success: response => console.log(response)
});
{readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}
```

✖ Access to XMLHttpRequest at '<http://localhost:3000/cors.php>' ([인덱스](#)):1 from origin '<https://jquery.com>' has been blocked by CORS policy: Method PUT is not allowed by Access-Control-Allow-Methods in preflight response.

# 특정 메서드만 – Access-Control-Allow-Methods

참고사항:

- 웹 서버 자체에서 메서드를 막으면 헤더를 줘도 못 쓴다

\* 대충 웹서버 설정이랑 CORS 에러나는 이미지



# 헤더좀 받아주세요 – Access-Control-Allow-Headers

- 때로는 요청에 헤더가 같이 붙어야 하기도 함  
e.g.) Content-Type, X-Requested-By, Authorization ...
- 아까까지의 페이지에 Authorization 헤더를 붙여 요청해보자

```
$origin = isset($_SERVER['HTTP_ORIGIN']) ? $_SERVER['HTTP_ORIGIN'] : null;  
header("Access-Control-Allow-Origin: $origin");  
header("Access-Control-Allow-Credentials: true");  
header("Access-Control-Allow-Methods: PUT");
```

# 헤더좀 받아주세요 – Access-Control-Allow-Headers

```
> jQuery.ajax({
  url: "http://localhost:15267/acah.php",
  method: "PUT",
  xhrFields: {
    withCredentials: true
  },
  headers: {
    Authorization: "12345"
  },
  success: response => console.log(response)
});
```

◀ {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}

✖ Access to XMLHttpRequest at 'http://localhost:15267/acah.php (index):1p' from origin 'https://jquery.com' has been blocked by CORS policy: Request header field authorization is not allowed by Access-Control-Allow-Headers in preflight response.

# 헤더좀 받아주세요 – Access-Control-Allow-Headers

- Access-Control-Allow-Headers 헤더를 추가

- 사용법:

```
Access-Control-Allow-Headers: Authorization // Authorization 허용  
Access-Control-Allow-Headers: Content-Type, X-Requested-By // 여러개  
Access-Control-Allow-Headers: * // 전부 다 허용
```

- 주의: 와일드카드(\*) 는 Authorization을 포함하지 않음!

# 헤더좀 받아주세요 – Access-Control-Allow-Headers

```
$origin = isset($_SERVER['HTTP_ORIGIN']) ? $_SERVER['HTTP_ORIGIN'] : null;  
header("Access-Control-Allow-Origin: $origin");  
header("Access-Control-Allow-Credentials: true");  
header("Access-Control-Allow-Methods: PUT");  
header("Access-Control-Allow-Headers: Authorization"); // Authorization을 허용하도록 추가
```

```
> jQuery.ajax({  
  url: "http://localhost:15267/acah.php",  
  method: "PUT",  
  xhrFields: {  
    withCredentials: true  
  },  
  headers: {  
    Authorization: "12345"  
  },  
  success: response => console.log(response)  
});  
< {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, se  
  tRequestHeader: f, overrideMimeType: f, ...}  
hi VM702:10
```

# 노출증 환자?



© tvN, CJ E&M Corporation

쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 노출증 환자?



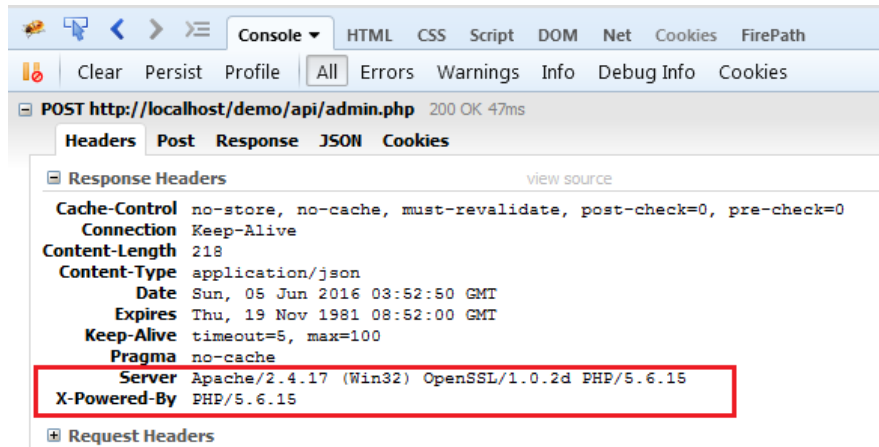
© 웨어하우스 @ YouTube

# 노출증 환자 – Server, X-Powered-By 헤더

- 서버들이 클라이언트에게 주로 전송하는 헤더들
- Server: 주로 웹서버나 엔드포인트 서버들이 전송  
Apache, Nginx, IIS, Lighttpd, ...
- X-Powered-By: WAS들이 주로 전송  
Tomcat, Kestrel(ASP.NET), PHP, Flask, Django, ...
- 이게 왜 문제?

# 노출증 환자 – Server, X-Powered-By 헤더

- 대부분 이런 헤더들은 쓸모가 없다
- 각 서버 데몬, 응용 프로그램의 버전이나
- 구동하고 있는 운영체제의 정보를 같이 전송하기도 함



<http://tanmaysarkar.com/remove-server-banner-details-from-response-header/>



<https://support.smartbear.com/readyapi/faq/how-to-write-file-stream-data-from-a-rest-response/>

쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------



# 노출증 환자 – Server, X-Powered-By 헤더

- 지피지기면 백전백승
  - 서버의 정보가 공격자에게 가장 쉽게 노출되는 경로
  - 즉, Banner Grabbing의 주요 경로
- 
- 각 서버 프로그램의 설정을 변경하면 삭제할 수 있음
    - 아니면 아예 소스를 바꿔서 컴파일 해도 된다. (버전 유지보수할 자신만 있다면야...)

# 노출증 환자 – Server, X-Powered-By 헤더

- Apache의 경우 – httpd.conf를 연 뒤

ServerTokens Prod

ServerSignature Off

로 변경하면 OS, Apache 버전 삭제 가능

- 하지만 Server 헤더가 완전히 삭제되지 않는다
- 만약 완전히 삭제하고 싶다면 modsecurity와 같은 WAF 설치

# 노출증 환자 – Server, X-Powered-By 헤더

- Nginx의 경우 – nginx.conf를 연 뒤 http나 server 섹션에

`server_tokens off;`

로 변경하면 버전 삭제 가능 (없으면 추가)

- 하지만 이 역시 Server 헤더가 완전히 삭제되지 않는다
- 완전히 삭제하고 싶다면 ngx\_security\_headers 등의 모듈 설치

# 노출증 환자 – Server, X-Powered-By 헤더

- PHP의 경우 – php.ini 를 연 뒤에

`expose_php = off`

- ASP.NET Core 의 경우 – Program.CreateHostBuilder() 에서

```
WebHost.CreateDefaultBuilder(args)
    .UseKestrel(c => c.AddServerHeader = false)
    .UseStartup<Startup>()
    .Build();
```

# 날 내버려두세요! - Do Not Track (DNT)

- 트래커?
- 사용자의 취향이나 습관, 페이지 접속정보 등을 관찰
- 쿠키, 스크립트 등등등...
- 주로 맞춤형광고 기능을 위해 사용

# 날 내버려두세요! - Do Not Track (DNT)

게임·라이프 > 게임·인터넷

## "구글이 24시간 당신의 목소리를 듣고 있다"...진실 논란 실제 해보니

차현아 기자



입력 2018.12.19 17:53

"구글은 당신과 당신 주변의 목소리를 다 듣고 있다. 구글은 마이크로 모든 음성을 데이터로 저장해놓고 있다."

최근 인터넷 커뮤니티를 중심으로 '어느 유튜버가 알아낸 구글에 대한 충격적인 진실'이라는 내용의 글이 떠돌았다. 이 글은 어느 유튜버가 "유튜브가 이용자 마이크로 들리는 모든 음성을 데이터로 기록하고 있다"는 내용을 실험한 결과를 보여준다.



### 3#

확인을 모두 마치고 유튜버는 크롬을 종료시킨 뒤 헤드셋 마이크를 켜 상태로 '강아지 장난감' 에대해서 말하기 시작하는데요,

'나는 강아지를 키우고 있는데 내 강아지를 위해서 강아지 장난감을 사줘야될거 같아, 그리고 내 강아지는 세상에서 최고이기 때문에 장난감을 사줄 것이고, 만약에 강아지 장난감을 구입하게된다면 kong(공) 장난감이나 plush(천)으로된 재질의 강아지 장난감을 사게된다면 좋겠어'

이런식으로 이야기를 하게됩니다,



### 4#

그리고 이야기를 마친 뒤에 다시 크롬을 실행시키고 다시 광고 배너가 포함되어있는 사이트를 방문해본 결과, 광고 배너에 '강아지 장난감' 에대한 광고 제품이 포함된 것을 보게되고 깜짝 놀라게 됩니다...

구글이 당신 컴퓨터의 마이크를 몰래 듣고있다는 것이 확인된 것입니다



쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 날 내버려두세요! - Do Not Track (DNT)



# 날 내버려두세요! - Do Not Track (DNT)



© NASA

쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------



# 날 내버려두세요! - Do Not Track (DNT)

- 클라이언트에서 서버로 전송하는 헤더
- 주로 DNT: 1 식으로 전송
- DNT 헤더가 수신될 경우 사용자에게 대한 추적을 중단해야 함  
서드파티 쿠키를 이용한 정보 수집, 외부 서비스 스크립트 회수 등등...
- 서비스의 보안이 중요하다면 **사용자의 사생활 또한 중요함**

쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 사이트 보안 점검하기

- 열심히 만든 사이트에 대한 보안 점검은 필수
  - 뚫리면 많이 슬프니까...
  - 특히 돈이 걸려 있는 사이트라 신뢰성이 중요하다면 더더욱
  - 사용한 라이브러리들이 보안에 취약한지도 알고 싶기도 하고...
- Banner Grabbing
- XSS, SQL Injection, Clickjacking, MIME Sniffing
- 이런 공격들에 취약한지 점검하는 방법은 없을까?

쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

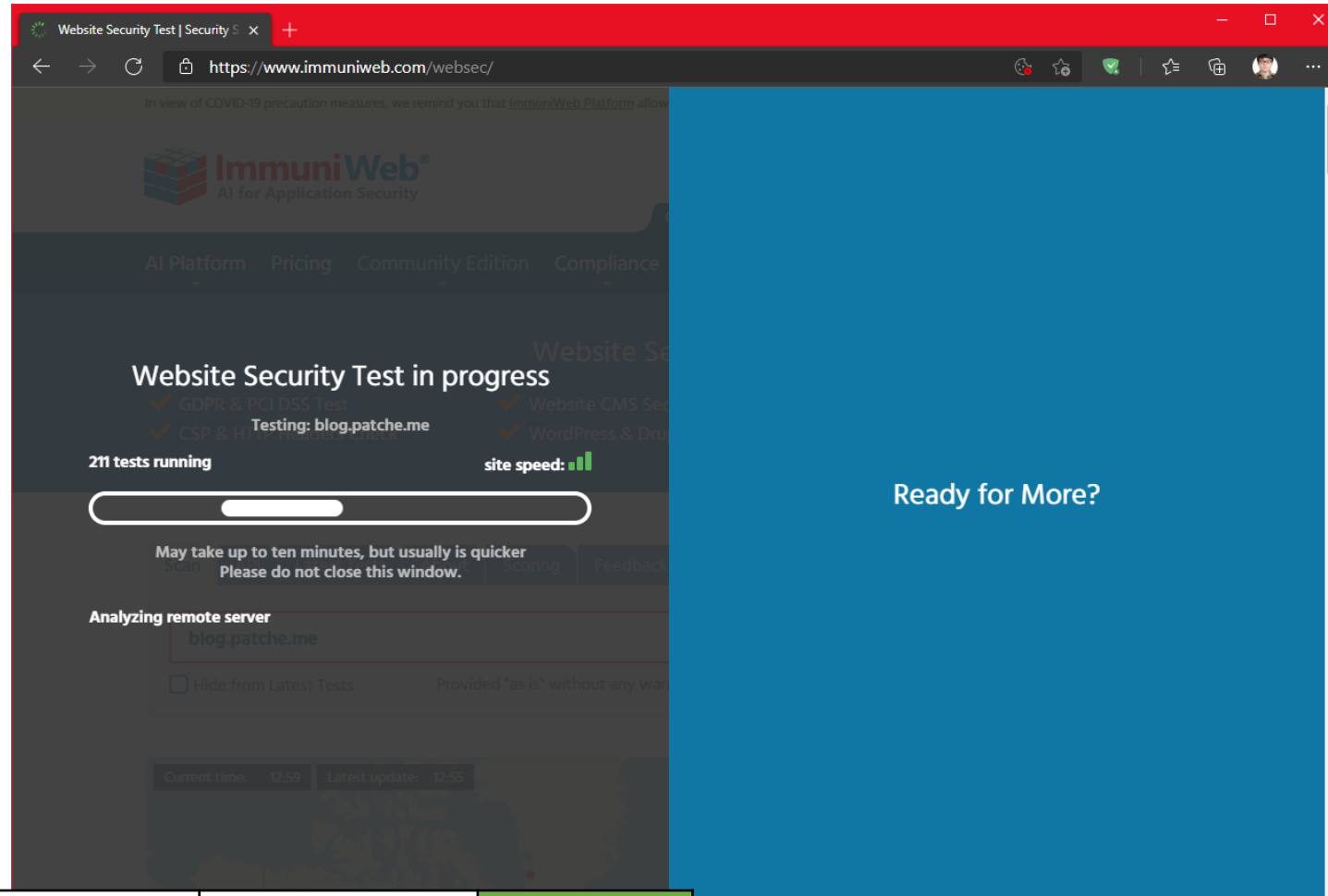
# 간단하게 점검해볼 수 있는 사이트 - Immuniweb

<https://www.immuniweb.com/websec/>

The screenshot displays the ImmuniWeb website interface. At the top, the ImmuniWeb logo is visible with the tagline "AI for Application Security". Navigation links include "Customer Login | Partner Login", "Community Edition", "Total Tests: 158,101,165", "This Week: 364,662", and "Today: 108,471". A dark blue navigation bar contains links for "AI Platform", "Pricing", "Community Edition", "Compliance", "Company", and "Partners", along with a "Get a Demo" button. The main section is titled "Website Security Test" and lists several services: "GDPR & PCI DSS Test", "Website CMS Security Test", "CSP & HTTP Headers Check", and "WordPress & Drupal Scanning". It also states "Free online security tool to test your security" and "57,292,172 security tests performed". Below this, there is a "Scan" section with tabs for "API", "Latest Tests", "About", "Scoring", and "Feedback". A search bar contains the URL "blog.patche.me" and a play button. To the right of the search bar, it shows "210 tests running" and "36,396 tests in 24 hours". A checkbox labeled "Hide from Latest Tests" is present, along with a disclaimer: "Provided 'as is' without any warranty of any kind". At the bottom, there is a "Current time: 12:58" and "Latest update: 12:55" section, a "View in fullscreen" button, and a map of the world.

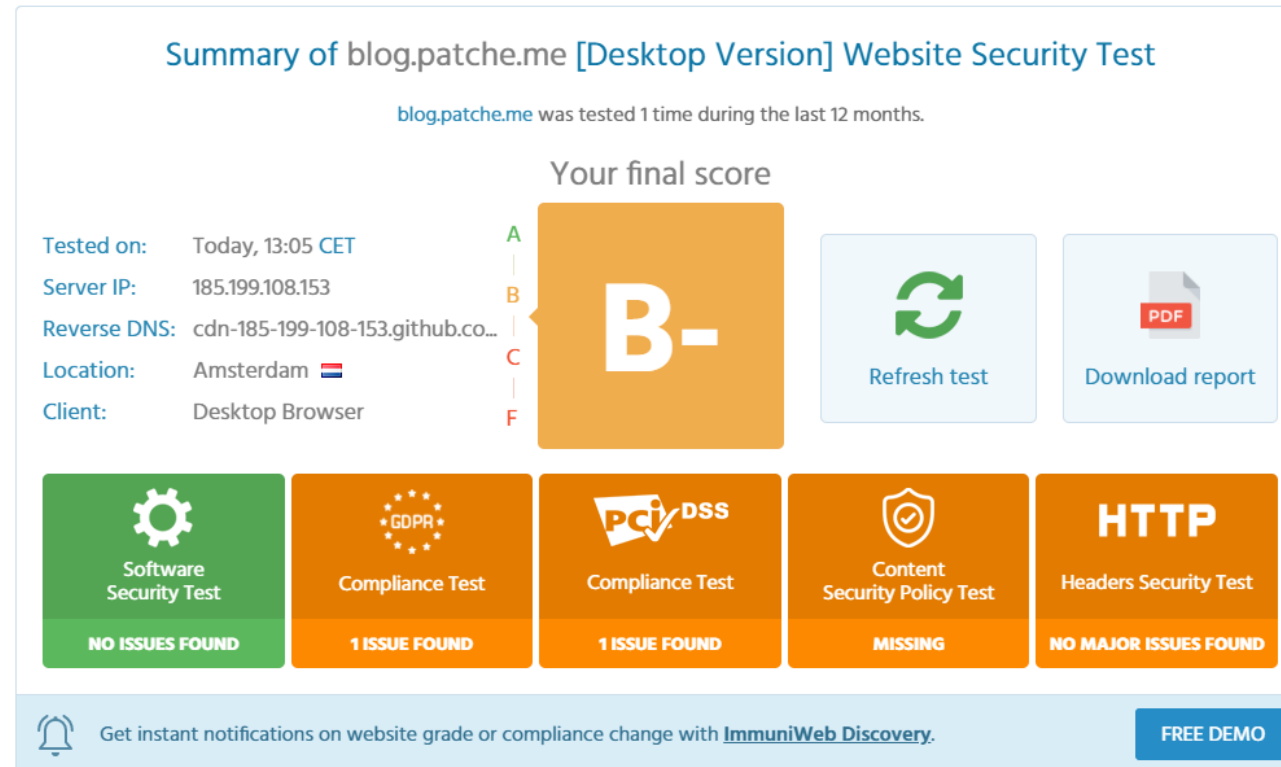
쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 간단하게 점검해볼 수 있는 사이트 - Immuniweb



쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 간단하게 점검해볼 수 있는 사이트 - Immuniweb



쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 간단하게 점검해볼 수 있는 사이트 - Immuniweb


## Web Server Security Test

<b>HTTP RESPONSE</b> 200 OK	<b>REDIRECT TO</b> N/A	<b>NPN</b> ⓘ N/A	<b>ALPN</b> ⓘ H2
<b>CONTENT ENCODING</b> GZIP	<b>SERVER SIGNATURE</b> GitHub.com	<b>WAF</b> ⓘ No WAF detected	<b>LOCATION</b> N/A
<b>HTTP METHODS ENABLED</b> ✔ GET ✔ HEAD	<b>HTTP REDIRECTS</b> ⓘ 1. http://blog.patche.me/ 2. https://blog.patche.me/		

 Get instant alerts about misconfigured or vulnerable web servers with [ImmuniWeb Discovery](#).[FREE DEMO](#)

# 간단하게 점검해볼 수 있는 사이트 - Immuniweb

## Software Security Test

 Get instant alerts about vulnerable or outdated web software with [ImmuniWeb Discovery](#).[FREE DEMO](#)

Web Software Found

1

Web Software Outdated

0

Web Software Vulnerabilities

0


### FINGERPRINTED CMS & VULNERABILITIES ⓘ

No CMS were fingerprinted on the website.[Information](#)

### FINGERPRINTED CMS COMPONENTS & VULNERABILITIES ⓘ

jQuery 3.5.1

The fingerprinted component version is up2date, no security issues were found.

 Get zero False Positives SLA testing and actionable remediation guidelines with [ImmuniWeb On-Demand](#).[FREE DEMO](#)

# 간단하게 점검해볼 수 있는 사이트 - Immuniweb

## HTTP Headers Security Test

Some HTTP headers related to security and privacy are missing or misconfigured. Misconfiguration or weakness

**MISSING REQUIRED HTTP HEADERS**

Strict-Transport-Security X-Frame-Options X-XSS-Protection X-Content-Type-Options

**MISSING OPTIONAL HTTP HEADERS**

Public-Key-Pins Public-Key-Pins-Report-Only Expect-CT Permissions-Policy

**SERVER**

Web server does not disclose its version. Good configuration

**Raw HTTP Header**


Server: GitHub.com

**ACCESS-CONTROL-ALLOW-ORIGIN**

The header is properly set. Good configuration

**Raw HTTP Header**

Access-Control-Allow-Origin: \*

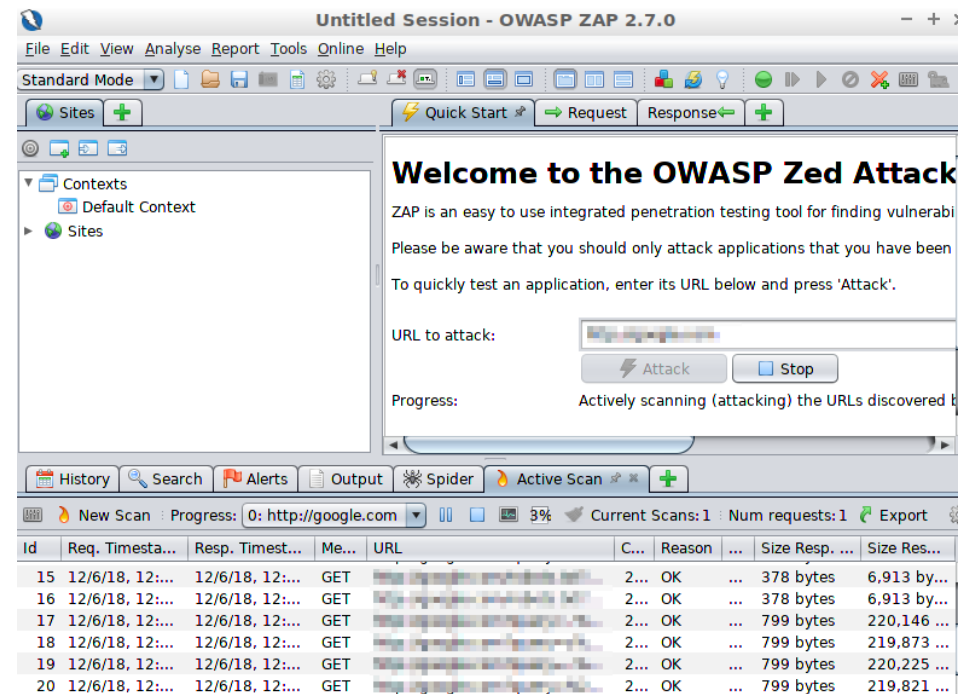
 Get continuous privacy and compliance monitoring with [ImmuniWeb Discovery](#). FREE DEMO

쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------



# 장비를 동원한 본격적인 점검 - ZAP

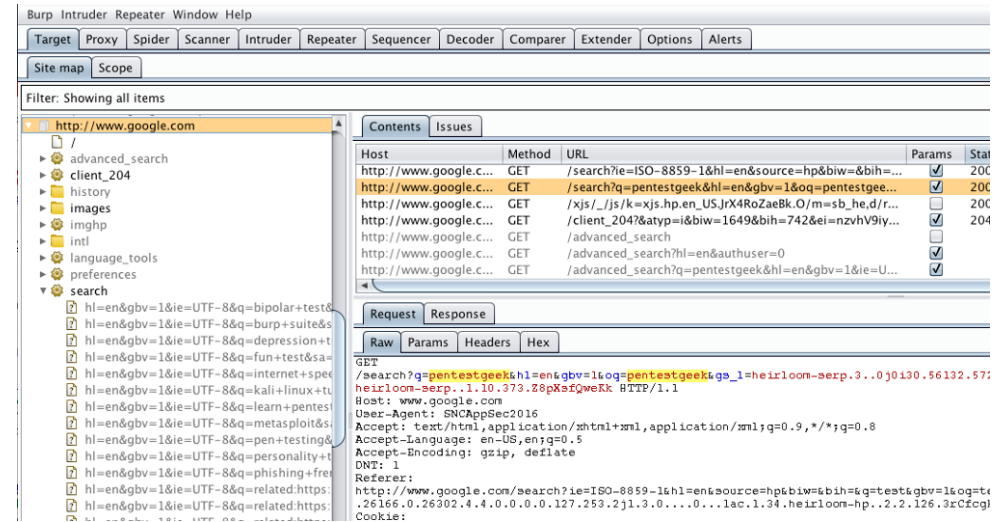
- OWASP에서 제작
- 무료!  
*가장 좋은 점은 무료라는 사실입니다*
- Windows, Linux, macOS 지원
- 마켓플레이스를 통한 애드온 확장
- <https://www.zaproxy.org/>



쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 사장님/교수님이 사주신다면? – Burp Suite

- PortSwigger에서 제작
- 무료버전은 Community Edition
- 하지만 좀 더 강력한 기능들을 편리하게 사용하고 싶다면
- 단돈 1년 \$399!
  - 비싸다



쿠키	헤더(넣을 것)	헤더(조심할 것)	점검도구
----	----------	-----------	------

# 사장님/교수님이 사주신다면? – Burp Suite

- ZAP나 Community Edition으로도 웬만한 경우는 다 커버됨
- 따라서 굳이 살 필요는 없다
- 하지만 정말 보안이 중요한 서비스거나
- 돈이 썩어나거나 윗선에서 사준다면 사서 써볼만 하다
- 앞의 immuniweb의 유료 서비스와 결합해볼 수도

# 정리하기

- 쿠키를 맛있게 구워서 대접하는 법
  - CSRF 막아주는 SameSite
  - XSS 막아주는 HttpOnly/Secure
  - 굳이 서버에서 알 필요가 없다면 localStorage/sessionStorage!
  - ...보안이 필요할때만 빼고! (굳이 필요하다면 암호화라도)
- 잘 알아두고 쓰면 좋은 헤더들
  - MITM/Downgrade Attack 막아주는 Strict-Transport-Security
  - XSS 막아주는 ~~X-XSS-Protection~~, Content-Security-Policy
  - Clickjacking 막아주는 X-Frame-Options
  - MIME Sniffing 막아주는 X-Content-Type-Option
  - API를 외부에 안전하게 공개 해야한다면 Access-Control-Origin 시리즈!

# 정리하기

- 주의합시다

- **Server, X-Powered-By** 헤더는 서비스 배포 전 삭제하기!
- 사용자가 **DNT**를 보낸다면 사생활을 존중해주세요.

- 배포전 점검해보기

- 공짜로 간단하게 쓰기 좋은 **immuniweb**
- 공짜지만 강력한 **ZAP**
- 좀 더 돈을 들여 백세게 테스트 하고 싶다면 **Burp Suite!**

# 사족: Web Application Firewall (WAF)

- 일반적인 네트워크 방화벽과는 다르다 (L7단 방어를 담당)
- 정책에 따라 XSS나 SQL Injection 등을 감지하고 차단



공짜지만 고생 깨나 해야하는 modsecurity

# 참고했던 / 할만한 자료들

- RFC6265: HTTP State Management Mechanism

쿠키의 스펙과 더불어 HTTP 프로토콜 상에서의 상태 유지에 관한 구체적인 표준 RFC 스펙이 담겨있습니다.

<https://tools.ietf.org/html/rfc6265>

- MDN(Mozilla) – Using HTTP Cookies

모질라재단에서 제공하는 HTTP Cookie 문서입니다.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

- Content Security Policy References

CSP 헤더에 관한 자세한 정보를 얻을 수 있는 레퍼런스 사이트입니다.

<https://content-security-policy.com/>

모질라에서 제공하는 자료도 참고해보세요.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>

# 참고했던 / 할만한 자료들

- OWASP: Cross Site Scripting Prevention Cheat Sheet

OWASP에서 제공하는 XSS 방어에 관한 팁 모음입니다.

[https://cheatsheetseries.owasp.org/cheatsheets/Cross\\_Site\\_Scripting\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)

- W3C Specification: Subresource Integrity

<https://www.w3.org/TR/SRI/>

- Mozilla MDN: X-Frame-Options

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>



# 참고했던 / 할만한 자료들

- OWASP: Cross Site Scripting Prevention Cheat Sheet

OWASP에서 제공하는 XSS 방어에 관한 팁 모음입니다.

[https://cheatsheetseries.owasp.org/cheatsheets/Cross\\_Site\\_Scripting\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)

- W3C Specification: Subresource Integrity

<https://www.w3.org/TR/SRI/>

- Mozilla MDN: X-Frame-Options

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>

# 참고했던 / 할만한 자료들

- Mozilla MDN: Access-Control-Allow-Origin, ...  
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin>

# 참고했던 / 할만한 자료들

발표 자료는 GitHub에 업로드 되어 있습니다!

<https://github.com/0x00000FF/DaechungCon-Safe-WebApp>

# Q&A

?!

